澳門大學
UNIVERSIDADE DE MACAU
UNIVERSITY OF MACAU

EMEN3000 Production Management

Course Project Report

HydroSense

ESP32-S3 Based Intelligent Plant Care System

Team1: sudo rain

DC327831 ZHU HAO LING
DC327955 CHENG WEI QI
DC326228 LIANG KIN CHON

Table of Contents

Abstract

Modern plant care often suffers from inconsistent watering patterns, either from neglect or over-watering, leading to poor plant health.

This project presents HydroSense, an ESP32-S3 based intelligent plant care system that addresses these challenges through automated soil moisture monitoring and precision watering control. The system features a four-layer software architecture (HAL/Managers/Services/UI), physical power gating via P-MOSFETs for ultra-low standby power consumption, and an innovative option-based dialogue interface that enables natural language interaction with an LLM assistant using only a rotary encoder input.

Key results demonstrate reliable automated watering with configurable thresholds, an intuitive e-ink display interface, and context-aware LLM responses that incorporate real-time sensor data.

The prototype achieves a hardware cost of approximately ¥186 with potential for significant cost reduction in volume production.

1. Introduction

1.1 Background and Motivation

Maintaining healthy houseplants presents a common challenge for many people. The difficulty lies not in the complexity of plant care itself, but in the consistency required—regular watering schedules are easily forgotten in busy modern lifestyles, while over-enthusiastic care can lead to over-watering and root rot. Traditional approaches either rely on human memory or provide only basic sensor readings without actionable guidance.

The proliferation of IoT devices and affordable microcontrollers has opened new possibilities for intelligent home automation. However, most existing smart plant monitors focus solely on data collection, leaving users to interpret readings and make watering decisions themselves. There remains a gap for systems that not only monitor but actively assist users with plant care in an intuitive, accessible manner.

An embedded solution offers distinct advantages for this application: long-term deployment capability, low maintenance requirements, and the ability to function offline for basic operations. These characteristics align well with the "set and forget" nature desired for plant care systems.

1.2 Project Goals and Requirements

The HydroSense project aims to create an intelligent plant care system with the following core objectives:

1. Automated monitoring of soil moisture with intelligent watering control based on configurable thresholds
2. Low-power design suitable for extended battery operation and unattended deployment
3. Intuitive local user interface via e-ink display and rotary encoder for direct interaction without requiring a smartphone
4. Optional LLM-powered "plant assistant" feature enabling natural language dialogue for plant care advice (network-dependent)

1.3 System Overview

HydroSense operates as a complete plant care solution with multiple input and output modalities:

Inputs: Capacitive soil moisture sensor, battery voltage monitoring, rotary encoder with push button, three-position mode switch

Processing: ESP32-S3 microcontroller running state machine logic, configuration management, WiFi networking, and LLM API integration

Outputs: Water pump control via MOSFET driver, 2.9" e-ink display for status and interaction, serial logging for debugging

The system operates in three distinct modes controlled by a physical switch:

- **OFF Mode:** Deep sleep with all peripherals powered down, minimal current draw
- **RUN Mode:** Autonomous operation with periodic sensor sampling and automatic watering
- **INTERACTIVE Mode:** Full user interface with menu navigation, settings adjustment, and LLM chat functionality

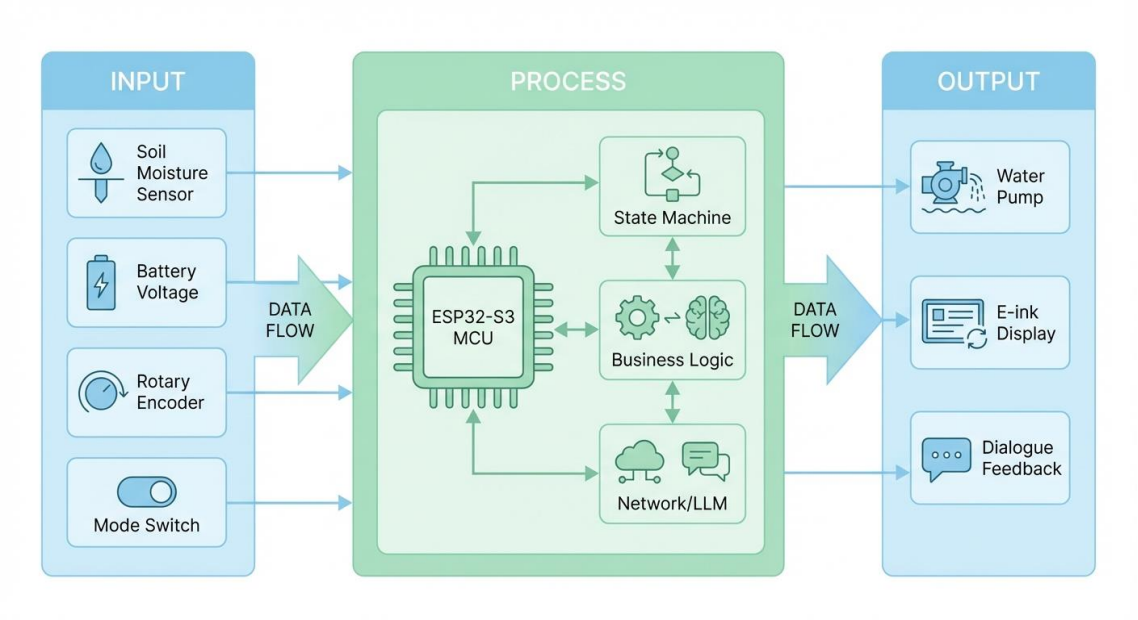

Figure 1: HydroSense System Block Diagram

1.4 Key Contributions and Highlights

This project makes several notable contributions:

- **MOSFET Physical Power Gating:** Hardware-level peripheral power control to minimize standby current consumption beyond what software sleep modes alone can achieve
- **Option-based LLM Dialogue:** Novel interaction paradigm enabling practical LLM conversations using only a single rotary encoder input
- **Layered Firmware Architecture:** Clean separation of concerns across HAL, Managers, Services, and UI layers for maintainability and extensibility
- **Physical Mode Switch:** Hardware-enforced operating modes providing deterministic behavior and fault tolerance without software dependencies

2. Detailed Design

2.0 Innovation Overview

Before diving into implementation details, this section highlights the key innovations that distinguish HydroSense from conventional plant monitoring solutions:

Innovation	Description
Hardware Low Power	P-MOSFET power gating + e-ink display (zero static power for display retention)
Hardware Reliability	Three-position physical mode switch ensuring deterministic operation
Software Architecture	Four-layer structure (HAL / Managers / Services / UI) for modularity
LLM Interaction	Option-based dialogue with structured JSON responses
Context Enhancement	Real-time sensor/config/history injection into LLM prompts
Development Efficiency	Serial CLI test mode with comprehensive hardware-in-loop testing

2.1 Overall Architecture

The HydroSense firmware follows a four-layer architecture that separates concerns and enables clean module boundaries:

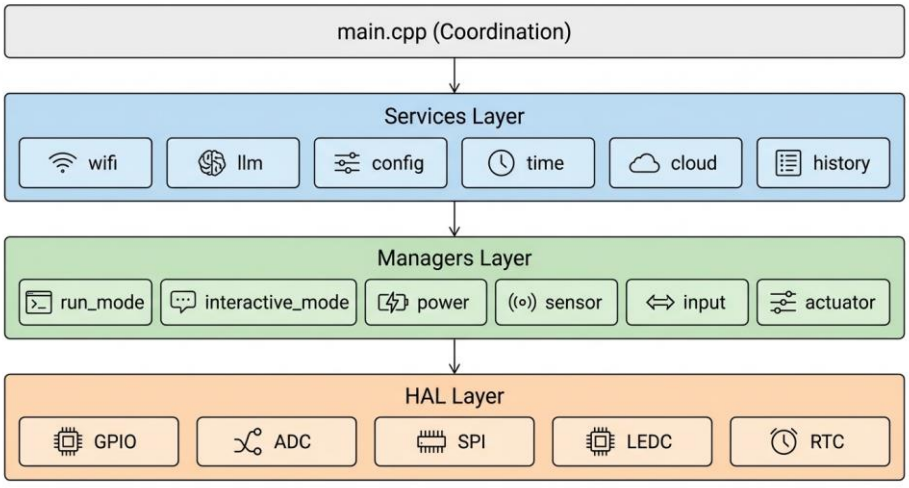

Figure 2: Four-Layer Software Architecture

Layer 1 - Hardware Abstraction (HAL): Provides low-level hardware interfaces including GPIO control, ADC sampling, SPI communication, LEDC PWM, and RTC management. All hardware-specific code is isolated here, making the upper layers hardware-agnostic.

Layer 2 - Managers: Implements business logic through dedicated managers: power_manager (peripheral power control), sensor_manager (measurement coordination), input_manager (encoder/switch handling), actuator_manager (pump control), and mode-specific managers (run_mode_manager, interactive_mode_manager).

Layer 3 - Services: High-level services including wifi_service (network connectivity), llm_connector (API communication), config_service (NVS persistence), time_service (NTP synchronization), and history_service (data logging).

Layer 4 - Coordination (main.cpp): Top-level coordination managing mode transitions, system initialization, and the main event loop. This layer remains thin, delegating most logic to the manager layer.

2.2 Operating Modes and State Machine

System behavior is governed by a three-state top-level state machine controlled by a physical switch:

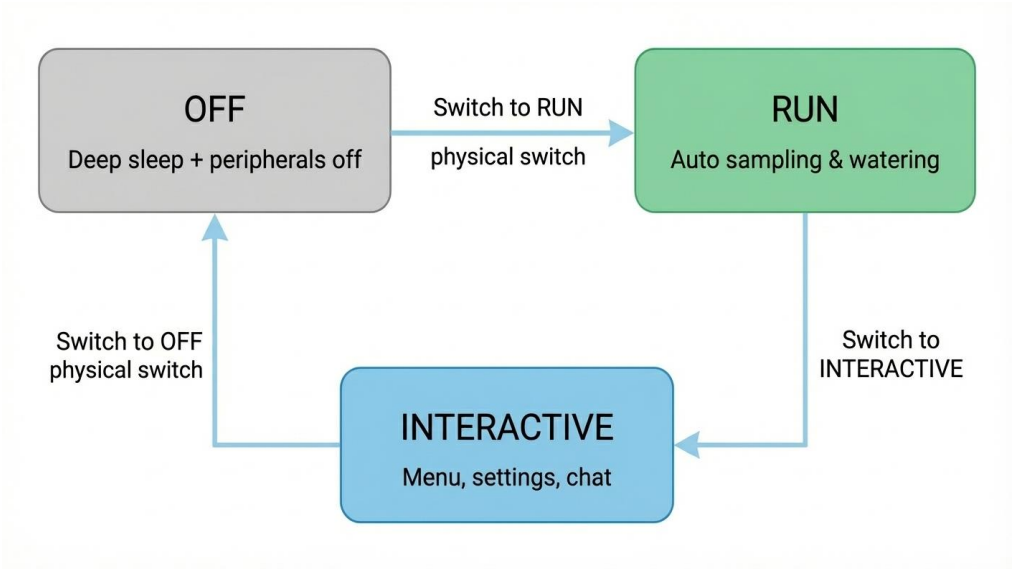

Figure 3: System Mode State Machine

OFF Mode

In OFF mode, the system enters ESP32-S3 deep sleep with all peripheral power gates disabled. Current consumption is minimized to the microcontroller's deep sleep current plus any unavoidable leakage. The system wakes only when the physical switch position changes, detected via external interrupt.

RUN Mode

RUN mode implements autonomous plant care with periodic wake cycles. Each cycle: powers sensors → waits for stabilization → samples moisture and battery → evaluates watering threshold → activates pump if needed → updates display → returns to light sleep. The run_mode_manager encapsulates all autonomous operation logic.

INTERACTIVE Mode

INTERACTIVE mode provides full user interface capabilities through a sub-state machine:

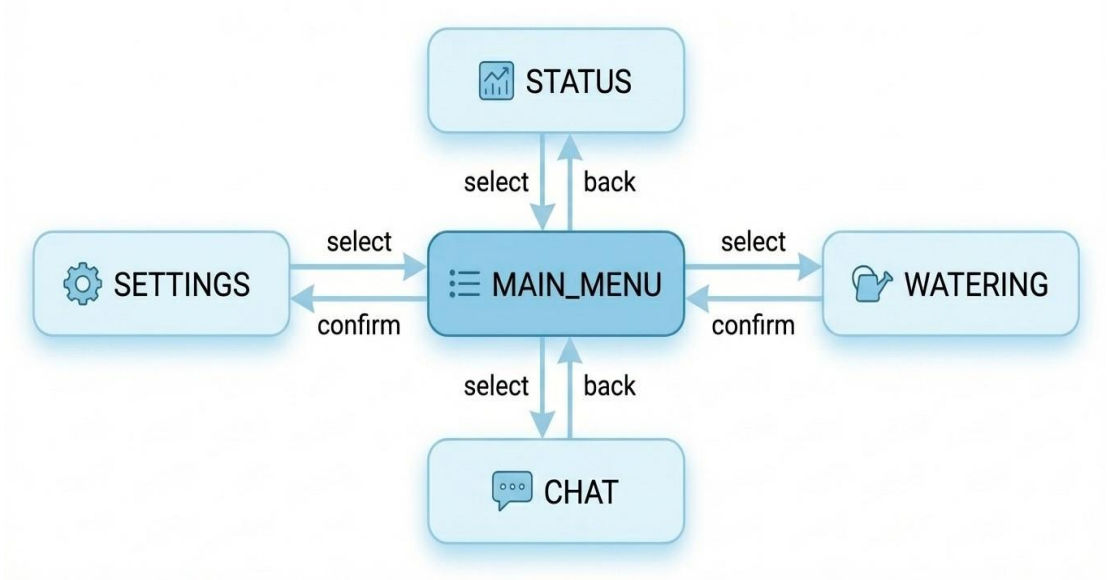

Figure 4: Interactive Mode Sub-States

- **Dashboard:** Main status display showing current moisture, battery level, and system state
- **Menu:** Navigation menu for accessing other functions
- **Settings:** Configuration interface for thresholds, WiFi, and LLM parameters

- **Watering:** Manual watering control with real-time feedback
- **Chat:** LLM-powered plant assistant dialogue interface

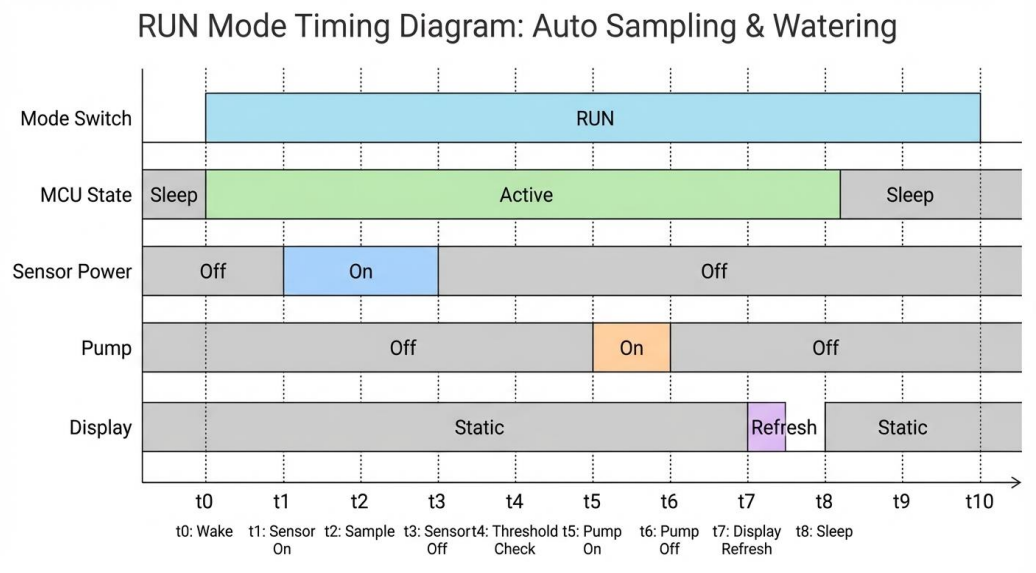

Figure 5: Event Timing Diagram (Wake/Sample/Water/Display)

2.3 Hardware Design

System Hardware Overview

The hardware design centers on the ESP32-S3 microcontroller with carefully selected peripherals for sensing, display, and actuation:

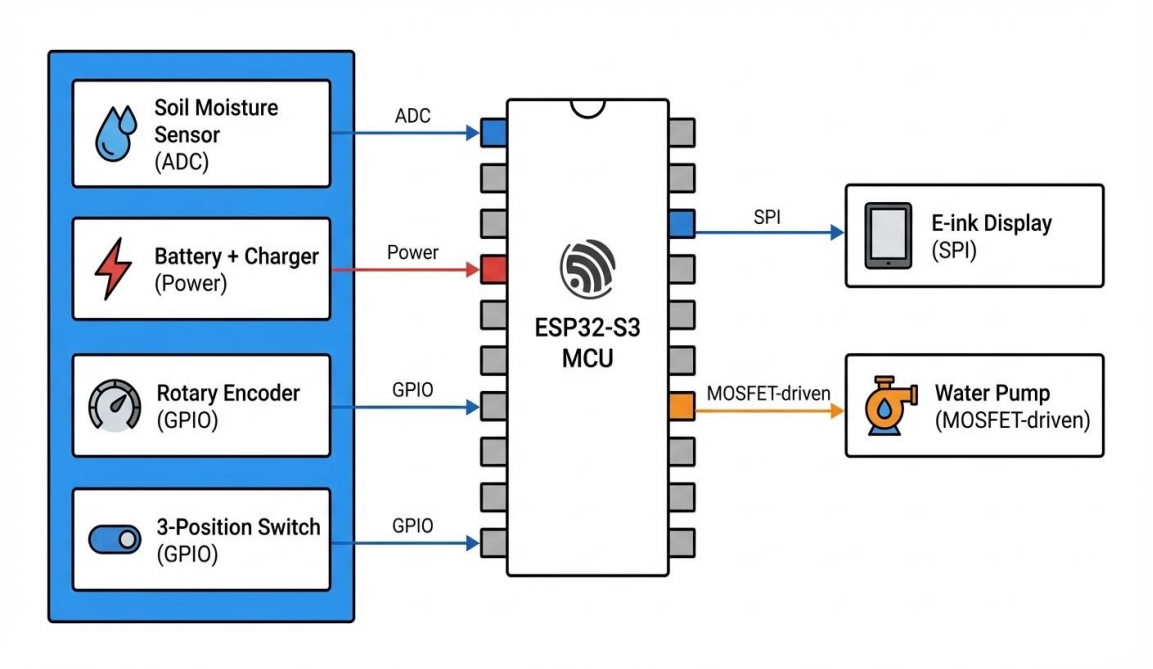

Figure 6: Hardware Block Diagram

Power Management and Gating

A key innovation is the use of P-channel MOSFETs as physical power gates for peripheral subsystems. This approach provides true hardware-level power isolation, eliminating leakage currents that would otherwise occur through powered-down but still connected peripherals. The power gating architecture includes separate gates for: sensor subsystem, display, and pump driver.

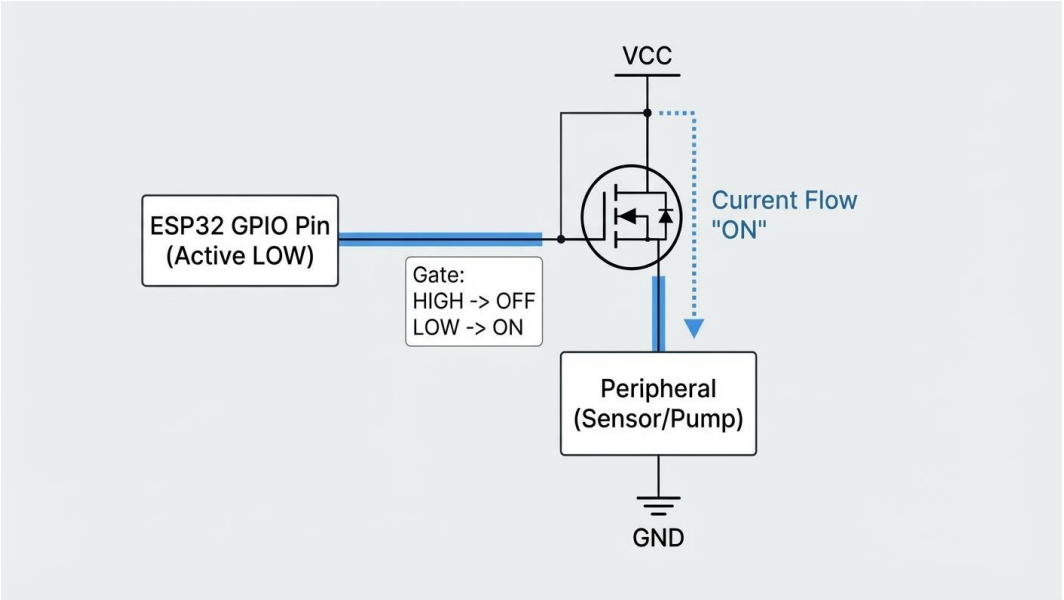

Figure 7: P-MOSFET Power Gating Architecture

All power gate control pins are active-LOW, allowing the ESP32-S3 to use internal pull-ups during deep sleep to keep peripherals disabled. Before entering sleep, GPIO pins are configured to prevent parasitic current paths through GPIO protection diodes.

Actuator and Safety Considerations

The water pump is driven through an N-channel MOSFET with flyback diode protection. Software safeguards include: maximum continuous run time limits, minimum interval between watering events, and dry-run detection. Physical measures include waterproof enclosure design and drainage provisions for the electronics compartment.

2.4 Software Implementation

Sensor Sampling Pipeline

Reliable sensor readings require a careful sequence: enable power gate → wait for sensor stabilization (typically 100ms for capacitive sensors) → perform multiple ADC samples → apply averaging/filtering → disable power gate. This approach ensures consistent readings while minimizing power consumption.

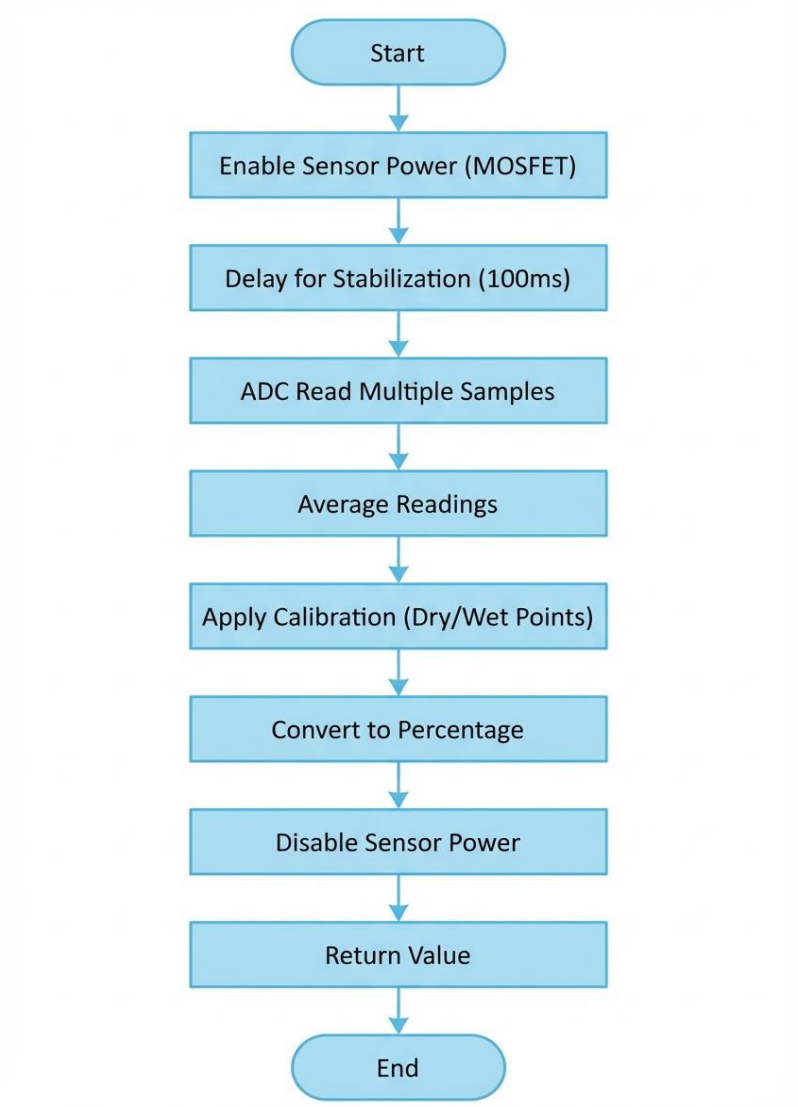

Figure 8: Sensor Sampling Flowchart

Calibration and Mapping

Raw ADC values are converted to percentage moisture using a two-point calibration: dry calibration point (sensor in air) and wet calibration point (sensor in saturated soil). Linear interpolation maps readings between these points to 0-100% moisture. Calibration values are stored in NVS for persistence.

Automatic Watering Strategy

The watering control algorithm implements hysteresis to prevent oscillation: watering triggers when moisture falls below a low threshold and continues until reaching a high threshold. Additional safeguards include minimum time between watering events (default 1 hour) and maximum daily watering limits.

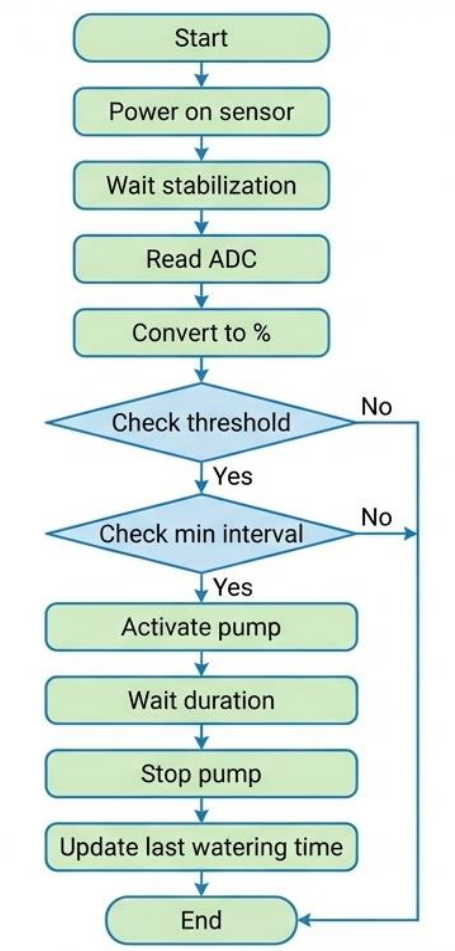

Figure 9: Watering Control Logic Flowchart

Configuration Persistence

System configuration is stored in ESP32's Non-Volatile Storage (NVS) partition. Stored parameters include: moisture thresholds, watering duration, WiFi credentials, LLM API configuration, calibration values, and user preferences. The config_service provides type-safe access with default fallback values.

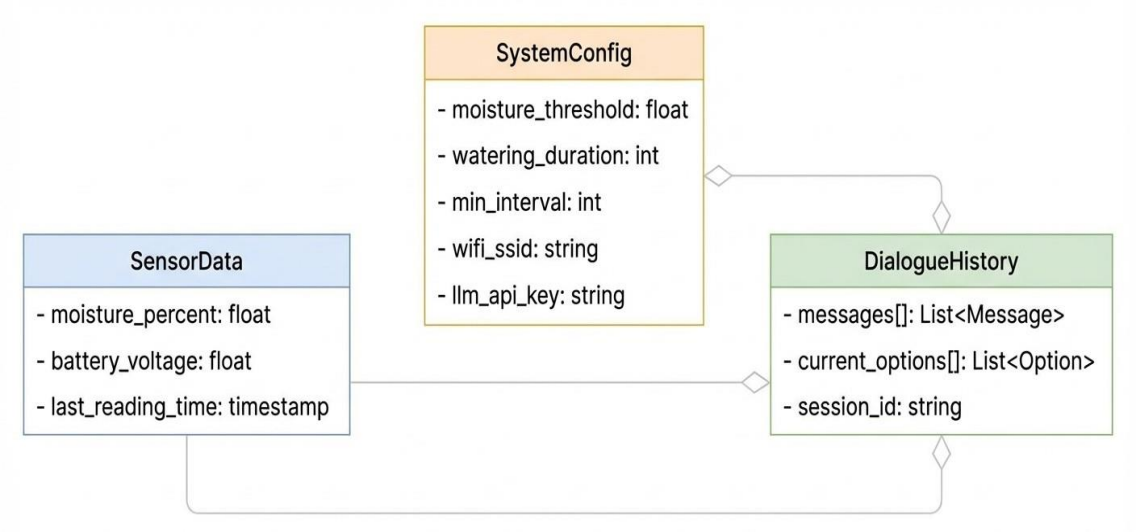

Figure 10: Key Data Structures

2.5 User Interface and Interaction

E-Ink Display Strategy

The 2.9" e-ink display provides excellent outdoor visibility and zero static power consumption for displayed content. The UI framework supports both full refresh (complete redraw, eliminates ghosting) and partial refresh (faster updates, some ghosting accumulation). Context-appropriate refresh modes balance responsiveness against display longevity.

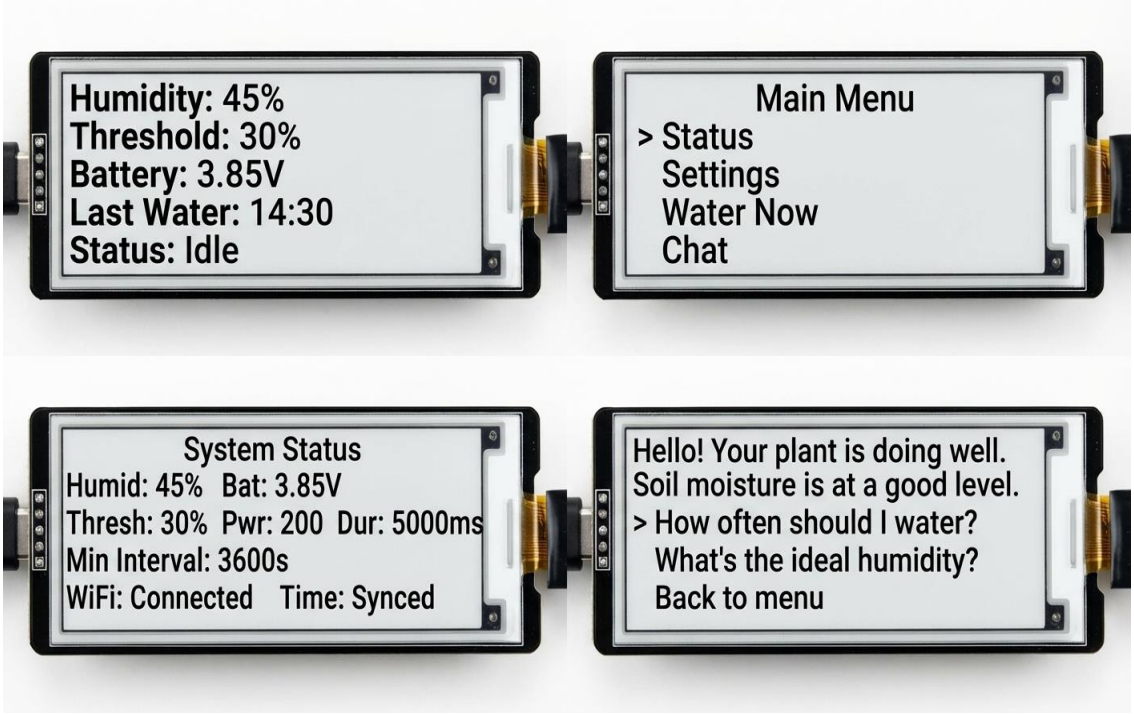

Figure 11: UI Screen Layouts (Dashboard, Menu, Settings, Chat)

Rotary Encoder Interaction

The rotary encoder provides the sole physical input mechanism (besides the mode switch). Interactions include: rotation for selection/value adjustment, short press for confirm/enter, and long press for back/cancel. Debouncing is handled in software with configurable timing parameters.

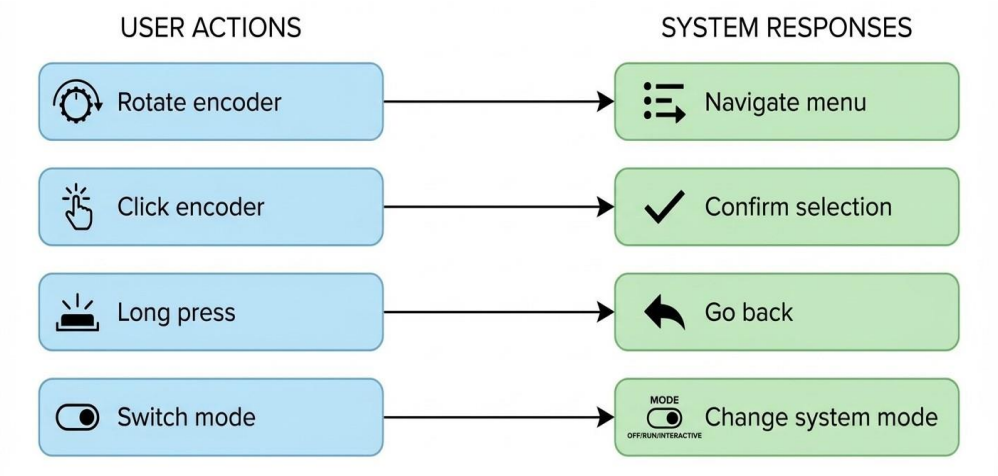

Figure 12: User Interaction Flow

2.6 LLM Integration (Highlight Feature)

Option-Based Dialogue Model

Traditional chat interfaces require text input, which is impractical with only a rotary encoder. HydroSense introduces an "option-based dialogue" paradigm: each LLM response includes both an answer and a set of suggested follow-up options. Users navigate responses by reading and select options by rotating and clicking—no typing required.

The LLM is instructed via system prompt to always return structured JSON with 'response' and 'options' fields. This ensures predictable parsing while allowing natural language content within the structure.

Context-Enhanced Prompts

To enable situationally-aware responses, each LLM request includes injected context: current sensor readings (moisture percentage, battery level), active configuration (thresholds, watering settings), current time, and optionally recent watering history. This allows the assistant to provide advice based on actual plant conditions rather than generic information.

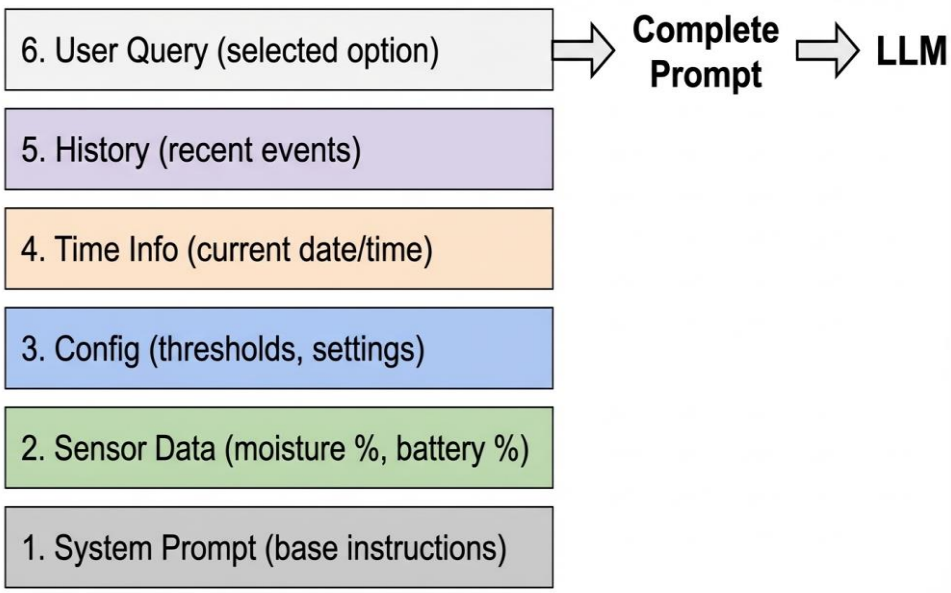

Figure 13: Context Injection Architecture

LLM Communication Pipeline

The end-to-end pipeline: UI gathers user selection → construct prompt with context → WiFi connection check → HTTPS POST to LLM API → receive response →

JSON parsing with fallback → display response and options → update UI state. Timeout handling and retry logic ensure graceful degradation on network issues.

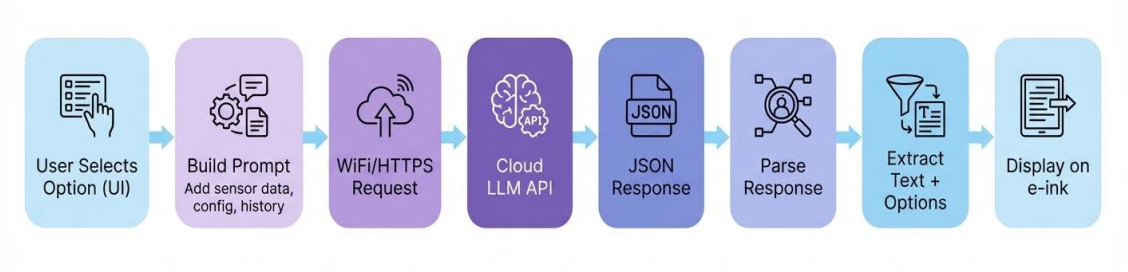

Figure 14: LLM Communication Pipeline

2.7 Testing and Debug Support

Serial CLI Test Mode

Building with the 'test' environment enables a comprehensive serial command-line interface for hardware-in-loop testing. Commands provide direct access to all hardware subsystems: sensor reading, display control, pump activation, WiFi testing, and LLM API verification. Test mode disables deep sleep to maintain serial connectivity.

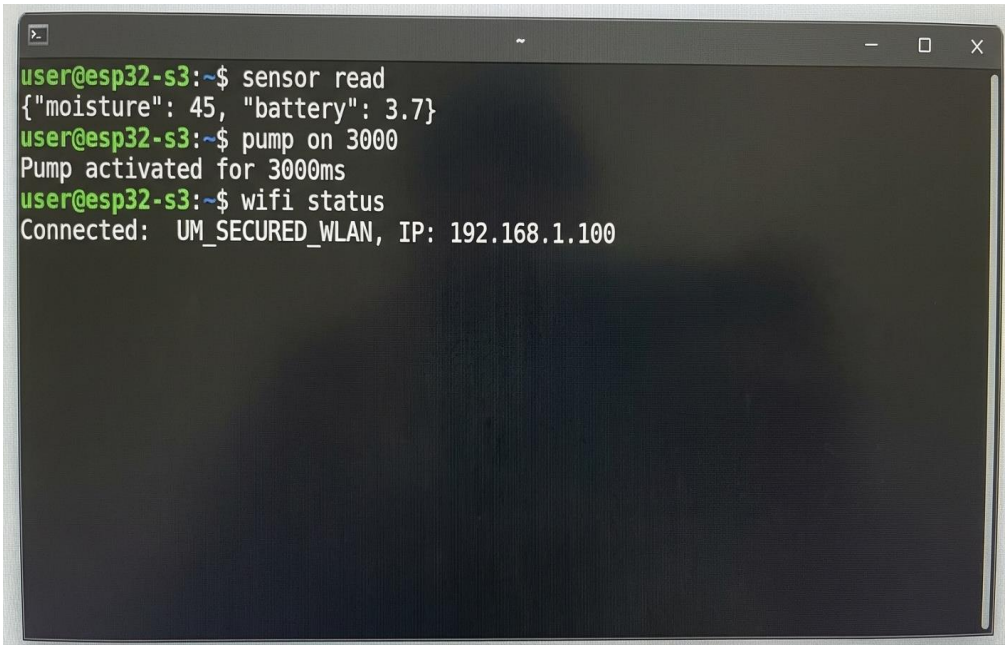

Figure 15: Serial CLI Test Interface

3. Results and Discussion

3.1 Implemented Features

The current prototype demonstrates the following operational capabilities:

1. **Automated Monitoring and Watering:** Periodic soil moisture sampling with threshold-based automatic pump control, including minimum interval protection
2. **E-Ink User Interface:** Dashboard display with moisture gauge, battery indicator, and status information; menu navigation for all functions
3. **WiFi Connectivity:** Configurable WiFi connection with stored credentials, connection status feedback
4. **LLM Chat Integration:** Option-based dialogue with context injection, conversation history support
5. **Persistent Configuration:** NVS-based storage for all user settings with defaults fallback
6. **Physical Mode Control:** Three-position switch providing hardware-level mode selection

3.2 Comparative Evaluation

Feature Comparison

Comparing HydroSense against traditional watering approaches and commercial smart plant monitors:

Feature	Traditional Pot	Basic Sensor	HydroSense
Auto watering	✗	✗	✓
Moisture display	✗	✓	✓
Low power sleep	✗	✗	✓
Local UI	✗	✗	✓
Voice/Chat assistant	✗	✗	✓
Offline capable	✗	✗	✓

Figure 16: Feature Comparison Matrix

Cost Analysis

The prototype bill of materials totals approximately ¥186, distributed across functional modules as shown:

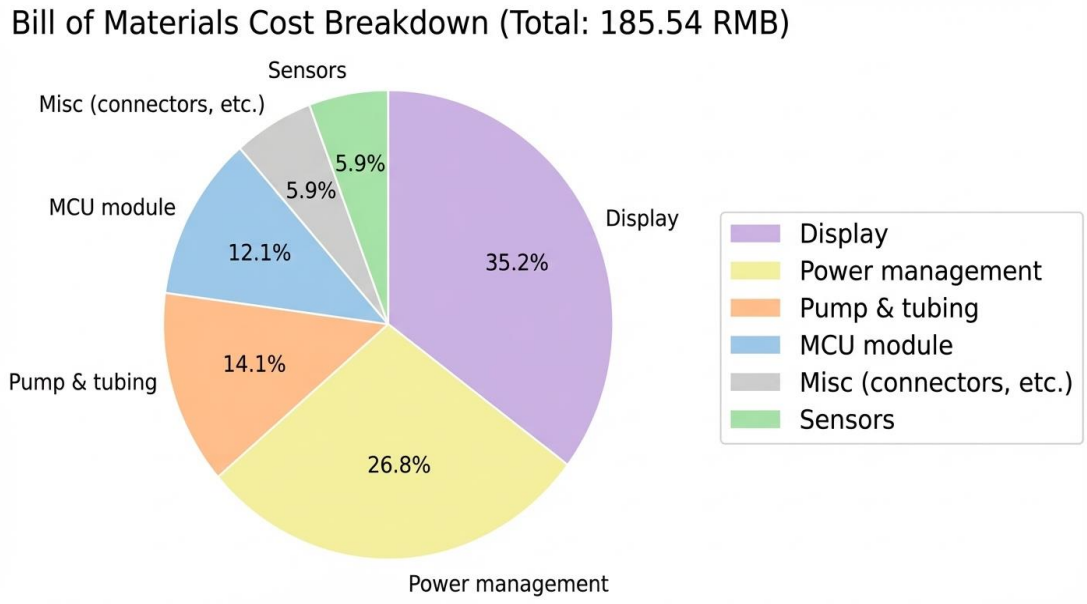

Figure 17: Prototype Cost Breakdown by Module

Volume production estimates suggest costs around ¥90 through: bulk component pricing, elimination of development board premiums (custom PCB instead of ESP32-S3-DevKitC), and component substitution where appropriate.

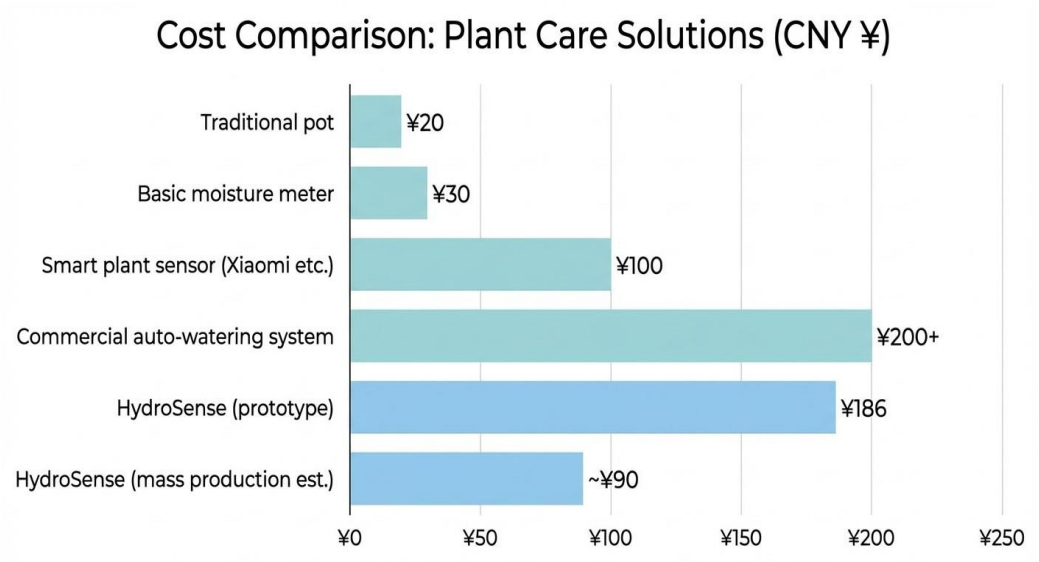

Figure 18: Cost Comparison with Commercial Products

User Experience Comparison

Traditional plant monitors display numerical readings that users must interpret. HydroSense's LLM integration transforms this interaction: rather than showing "Moisture: 23%", users can ask "Should I water my plant?" and receive contextual advice like "Your soil moisture is getting low at 23%. Based on the current reading and your last watering 3 days ago, I'd recommend watering soon."

3.3 Performance, Power, and Battery Life

Response Latency

UI operations exhibit the following typical latencies: e-ink full refresh ~2s, partial refresh ~0.3s, LLM round-trip ~3-8s depending on network conditions and response length. User input response is effectively instantaneous (<50ms).

Power Budget Analysis

Power consumption varies significantly by operating state:

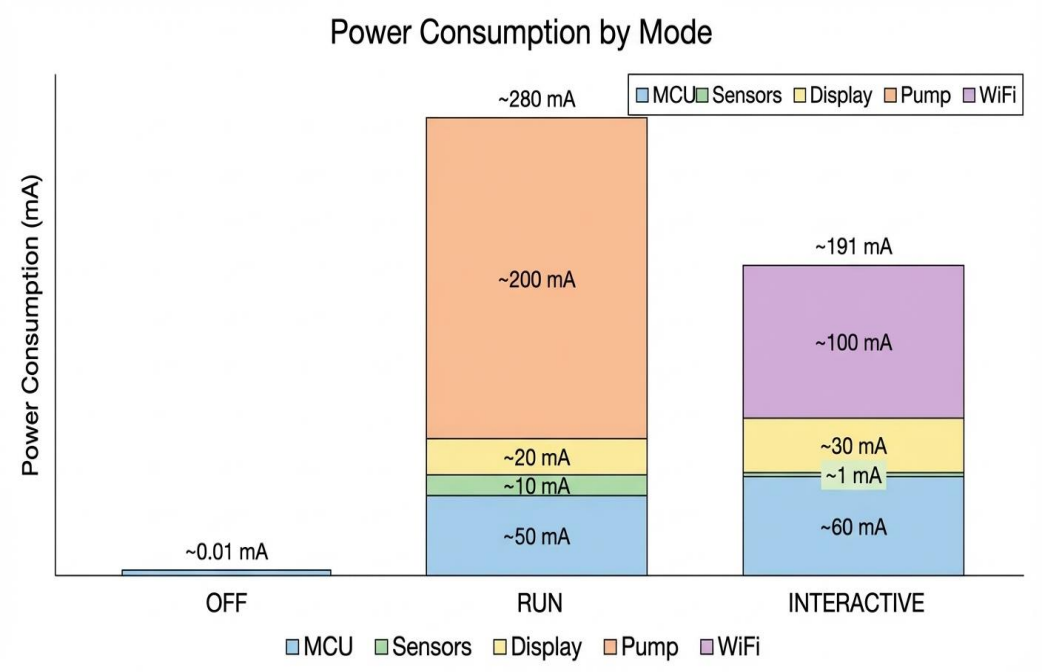

Figure 19: Power Budget by Operating State

Battery Life Estimation

Based on a typical usage model (hourly sampling in RUN mode, 10 interactive sessions per day averaging 2 minutes each, one watering event every 3 days), battery life projections for a 5000mAh LiPo cell:

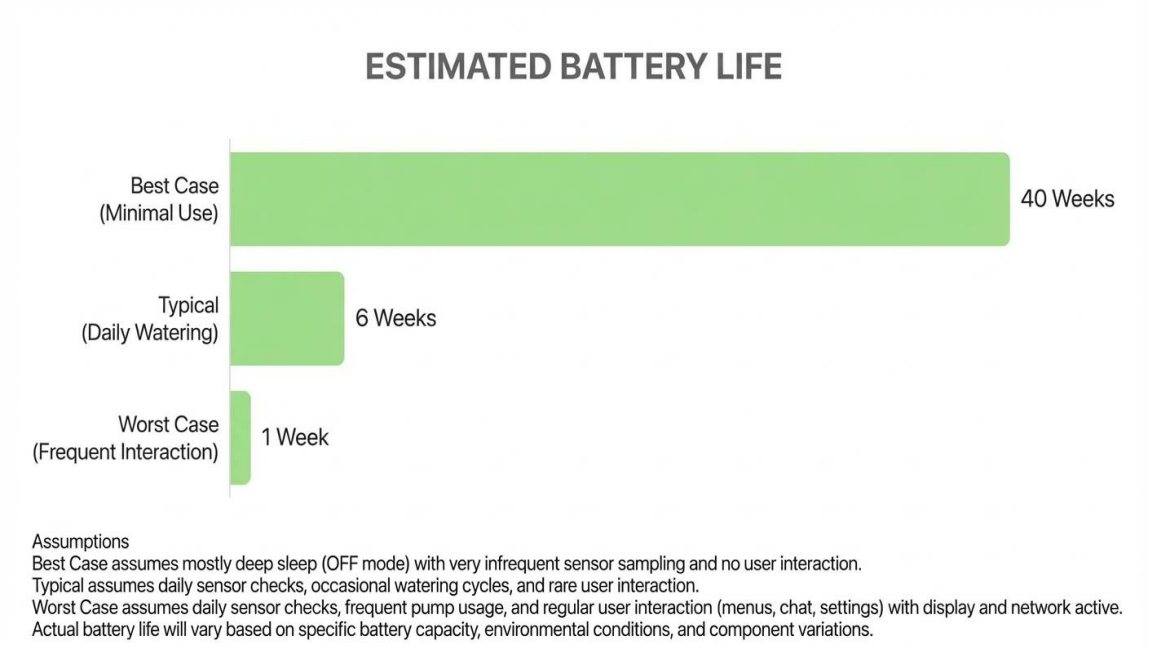

Figure 20: Battery Life Projection (Best/Typical/Worst Case)

Work Cycle Visualization

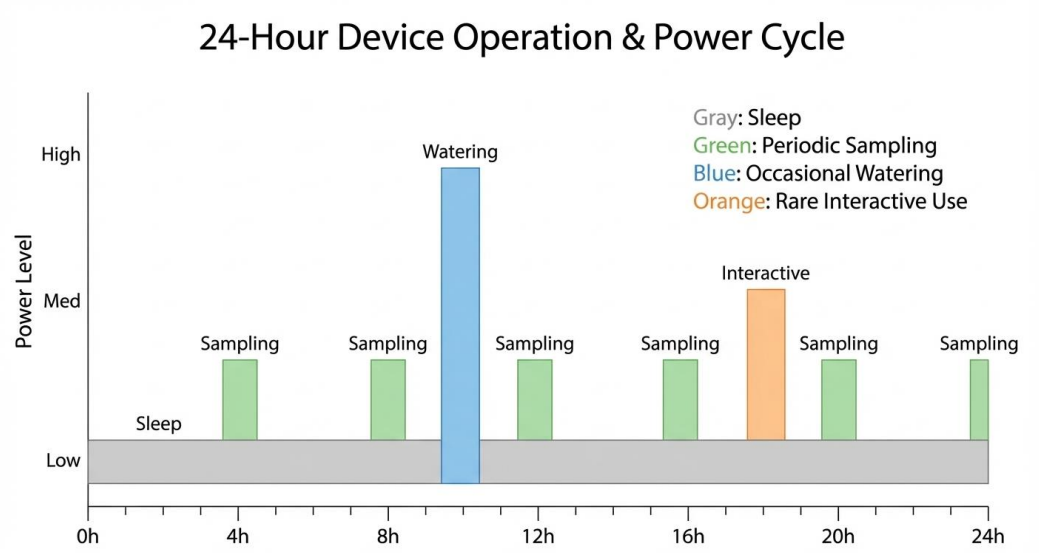

Figure 21: Typical Work Cycle Timeline

4. Conclusion

HydroSense successfully demonstrates an integrated approach to intelligent plant care, combining embedded sensing, automated actuation, and LLM-powered interaction in a cohesive system. The key achievements include:

- Reliable automated watering based on soil moisture monitoring with configurable thresholds and safety limits
- Hardware power gating architecture that minimizes standby power consumption
- Intuitive local interface via e-ink display and rotary encoder, eliminating smartphone dependency for basic operations
- Novel option-based dialogue paradigm enabling practical LLM interaction with minimal input hardware
- Clean four-layer firmware architecture supporting maintainability and future extension

The project required careful trade-offs given time constraints. Certain planned features were deferred, including ULP-based ultra-low-power RUN mode and cloud data synchronization.

Future Work

Potential directions for future development include:

1. **ULP Coprocessor Integration:** Implement "sentinel-commander" architecture where the Ultra-Low-Power coprocessor handles periodic monitoring, waking the main CPU only when intervention is needed
2. **Enhanced Robustness:** Comprehensive error handling, hardware watchdog integration, and graceful degradation strategies
3. **Cloud Connectivity:** Optional cloud data logging, remote monitoring dashboard, and companion mobile application
4. **Multi-Plant Support:** Mesh networking for coordinating multiple HydroSense units, enabling whole-garden monitoring

References

[1] Espressif Systems, "ESP32-S3 Technical Reference Manual," Version 1.6, 2025. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_technical_reference_manual_en.pdf

[2] Espressif Systems, "ESP32-S3 Datasheet," Version 1.7, 2025. [Online]. Available: https://www.espressif.com/sites/default/files/documentation/esp32-s3_datasheet_en.pdf

[3] Espressif Systems, "ESP-IDF Programming Guide," Version 5.1, 2025. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32s3/

[4] Espressif Systems, "ESP32-S3-DevKitC-1 User Guide," 2025. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32s3/hw-reference/esp32s3/user-guide-devkitc-1.html

[5] Espressif Systems, "Arduino-ESP32 Core Documentation," Version 3.x, 2025. [Online]. Available: https://docs.espressif.com/projects/arduino-esp32/en/latest/

[6] PlatformIO, "Espressif 32 Platform Documentation," 2025. [Online]. Available: https://docs.platformio.org/en/latest/platforms/espressif32.html

[7] Espressif Systems, "Sleep Modes," in *ESP-IDF Programming Guide*, Version 5.1, 2025. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32s3/api-reference/system/sleep_modes.html

[8] Espressif Systems, "Non-volatile Storage Library," in *ESP-IDF Programming Guide*, Version 5.1, 2025. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32s3/api-reference/storage/nvs_flash.html

[9] Espressif Systems, "Wi-Fi API Reference," in *ESP-IDF Programming Guide*, Version 5.1, 2025. [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v5.1/esp32s3/api-reference/network/esp_wifi.html

[10] Good Display, "GDEY029T94 2.9-inch E-ink Display," 2025. [Online]. Available: https://www.good-display.com/product/389.html

[11] Good Display, "GDEY029T94 Specification Document," 2022. [Online]. Available: https://www.good-display.com/companyfile/621.html

[12] J. M. Zingg, "GxEPD2 - Arduino Display Library for SPI E-Paper Displays," *GitHub*, 2025. [Online]. Available: https://github.com/ZinggJM/GxEPD2

[13] Alpha & Omega Semiconductor, "AO3401A 30V P-Channel MOSFET Datasheet," 2019. [Online]. Available: https://www.aosmd.com/pdfs/datasheet/AO3401A.pdf

[14] onsemi, "2N7002 / NDS7002A N-Channel Enhancement Mode FET Datasheet," 2023. [Online]. Available: https://www.onsemi.com/download/data-sheet/pdf/nds7002a-d.pdf

[15] NanJing Top Power ASIC Corp., "TP4056 1A Standalone Linear Li-Ion Battery Charger Datasheet," 2018. [Online]. Available: https://datasheet.lcsc.com/lcsc/1809261820_TOPPOWER-Nanjing-Extension-Microelectronics-TP4056-42-ESOP8_C16581.pdf

[16] Shanghai Consonance Electronics, "CN3795 4A Multi-Chemistry Battery Charger with MPPT Datasheet," 2020. [Online]. Available: https://www.mikrocontroller.net/attachment/609796/CN3795-CONSONANCE.pdf

[17] Torex Semiconductor, "XC6206 Series Ultra Low Power LDO Voltage Regulator Datasheet," 2023. [Online]. Available: https://product.torexsemi.com/system/files/series/xc6206.pdf

[18] DFRobot, "Capacitive Soil Moisture Sensor SKU:SEN0193 Wiki," 2023. [Online]. Available: https://wiki.dfrobot.com/Capacitive_Soil_Moisture_Sensor_SKU_SEN0193

[19] B. Blanchon, "ArduinoJson," Version 7, 2025. [Online]. Available: https://arduinojson.org/v7/

[20] OpenAI, "API Reference," 2025. [Online]. Available: https://platform.openai.com/docs/api-reference

[21] OpenAI, "Chat Completions Guide," 2025. [Online]. Available: https://platform.openai.com/docs/guides/chat-completions

[22] IETF, "RFC 5905 - Network Time Protocol Version 4: Protocol and Algorithms Specification," 2010. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc5905

Appendix

A. System Hardware Schematic

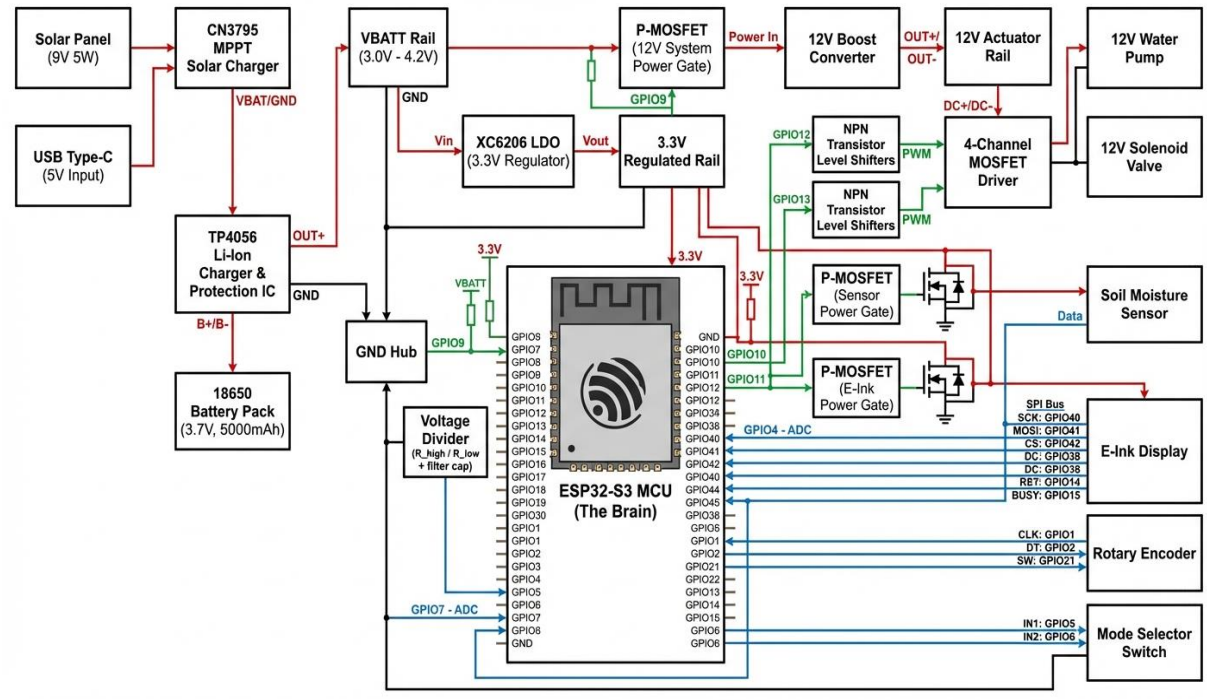

Figure A1: Complete System Hardware Schematic

B. Key Code Snippets

B.1 P-MOSFET Power Gating Implementation

Source: *src/managers/power_manager.cpp*

Physical power control using P-channel MOSFETs. All power gates are active-LOW, meaning POWER_OFF = HIGH and POWER_ON = LOW. This design enables complete peripheral shutdown for ultra-low standby power consumption.

```
power_result_t power_manager_init() {
    // 1. Initialize all power gate pins as outputs
    hal_gpio_pin_mode(PIN_POWER_GATE_PUMP, OUTPUT);
    hal_gpio_pin_mode(PIN_POWER_GATE_SENSOR, OUTPUT);
    hal_gpio_pin_mode(PIN_POWER_GATE_DISPLAY, OUTPUT);

    // 2. Force all peripherals OFF (P-MOSFET: HIGH = OFF)
    hal_gpio_write(PIN_POWER_GATE_PUMP, POWER_OFF);
```

```
    hal_gpio_write(PIN_POWER_GATE_SENSOR, POWER_OFF);
    hal_gpio_write(PIN_POWER_GATE_DISPLAY, POWER_OFF);

    // 3. Sync software state with hardware
    is_pump_module_powered = false;
    is_sensor_powered = false;
    is_screen_powered = false;
    is_initialized = true;

    return POWER_OK;
}

power_result_t power_sensor_enable(bool enable) {
    if (!is_initialized) return POWER_ERROR_NOT_INIT;
    if (is_sensor_powered == enable) return POWER_OK;  // No-op if
unchanged

    hal_gpio_write(PIN_POWER_GATE_SENSOR, enable ? POWER_ON : POWER_OFF);
    is_sensor_powered = enable;
    return POWER_OK;
}
```

B.2 Deep Sleep Entry Sequence

Source: *src/main.cpp + src/hal/hal_rtc.cpp*

Complete shutdown sequence before entering ESP32-S3 deep sleep mode. The system displays a shutdown screen, disables all peripherals, and configures EXT1 wakeup on the mode switch pin transition.

```
// main.cpp - Shutdown sequence
static void enter_off_mode_logic() {
    LOG_INFO("Main", "Entering OFF mode...");

    // 1. Stop all actuators
    actuator_manager_stop_pump();

    // 2. Show shutdown screen
    power_screen_enable(true);
    ui_manager_show_shutdown_screen();
    delay(200);  // Wait for e-ink refresh

    // 3. Disable all peripheral power
    power_sensor_enable(false);
```

```
    power_pump_module_enable(false);
    power_screen_enable(false);

    // 4. Enter deep sleep
    hal_rtc_enter_deep_sleep();
}

// hal_rtc.cpp - Deep sleep configuration
void hal_rtc_enter_deep_sleep() {
    // Wake on mode switch: OFF->RUN or OFF->INTERACTIVE
    // OFF: A=LOW, B=HIGH | RUN: A=HIGH, B=LOW | INTERACTIVE: A=HIGH,
B=HIGH
    // Leaving OFF means pin A goes LOW->HIGH
    esp_sleep_enable_ext1_wakeup(
        (1ULL << PIN_MODE_SWITCH_A),
        ESP_EXT1_WAKEUP_ANY_HIGH
    );

    // Disable RTC peripherals for minimum power consumption
    esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_PERIPH, ESP_PD_OPTION_OFF);
    esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_SLOW_MEM, ESP_PD_OPTION_OFF);
    esp_sleep_pd_config(ESP_PD_DOMAIN_RTC_FAST_MEM, ESP_PD_OPTION_OFF);

    esp_deep_sleep_start();  // Never returns
}
```

B.3 Option-Based LLM Response Parsing

Source: *src/services/llm_connector.cpp*

Parses structured JSON responses from the LLM API. The system prompts the LLM to return both a text response and an array of conversation options, enabling rotary encoder-based dialogue selection on the embedded device.

```
bool LLMConnector::parseStructuredResponse(
    const String& response_json,
    char* response_buffer, size_t response_size,
    char options[][64], uint8_t* option_count)
{
    // Parse outer API response
    JsonDocument api_doc;
    deserializeJson(api_doc, response_json);
    const char* content = api_doc["choices"][0]["message"]["content"];

    // Parse inner JSON structure from LLM content
```

```
    // Expected format: {"response": "...", "options": ["opt1", "opt2",
"opt3"]}
    JsonDocument content_doc;
    DeserializationError error = deserializeJson(content_doc, content);

    if (!error) {
        // Extract response text
        const char* response_text = content_doc["response"];
        strncpy(response_buffer, response_text, response_size - 1);

        // Extract options array
        JsonArray options_array = content_doc["options"].as<JsonArray>();
        *option_count = 0;
        for (JsonVariant option : options_array) {
            if (*option_count >= 3) break;
            strncpy(options[*option_count], option.as<const char*>(),
63);
            (*option_count)++;
        }
    }

    // Fallback options if LLM didn't provide any
    if (*option_count == 0) {
        strncpy(options[0], "Let's keep chatting", 63);
        strncpy(options[1], "View sensor data", 63);
        strncpy(options[2], "Meow!", 63);
        *option_count = 3;
    }
    return true;
}
```

B.4 Interactive Mode State Machine

Source: *src/managers/interactive_mode_manager.cpp*

Central coordinator for the interactive user interface. Routes input events to specialized state handlers and manages state transitions with proper enter/exit lifecycle calls.

```
interactive_mode_result_t interactive_mode_manager_loop(void) {
    if (!is_initialized) return INTERACTIVE_MODE_ERR_NOT_INITIALIZED;

    interactive_state_t prev_state = current_state;

    // Route to appropriate state handler
```

```
    switch (current_state) {
        case STATE_MAIN_MENU:
            interactive_main_menu_handle(&current_state,
&exit_requested);
            break;
        case STATE_STATUS:
            interactive_status_handle(&current_state);
            break;
        case STATE_SETTINGS:
            interactive_settings_handle(&current_state);
            break;
        case STATE_SETTING_EDIT:
            interactive_setting_edit_handle(&current_state);
            break;
        case STATE_WATERING:
            interactive_watering_handle(&current_state);
            break;
        case STATE_CHAT:
            interactive_chat_handle(&current_state);
            break;
        default:
            current_state = STATE_MAIN_MENU;
            break;
    }

    // Call enter() when state changes
    if (current_state != prev_state) {
        switch (current_state) {
            case STATE_MAIN_MENU:   interactive_main_menu_enter();
break;
            case STATE_STATUS:      interactive_status_enter();
break;
            case STATE_SETTINGS:    interactive_settings_enter();
break;
            case STATE_SETTING_EDIT: interactive_setting_edit_enter();
break;
            case STATE_WATERING:    interactive_watering_enter();
break;
            case STATE_CHAT:        interactive_chat_enter();
break;
        }
    }
    return INTERACTIVE_MODE_OK;
}
```

B.5 Rotary Encoder Polling with FreeRTOS

Source: *src/managers/input_manager.cpp*

High-priority FreeRTOS task for rotary encoder state decoding using a lookup table. Events are accumulated until reaching a threshold, then queued for consumption by the main loop. Critical sections protect shared state from race conditions.

```
static void encoder_polling_task(void* parameter) {
    // Gray code state transition lookup table
    static const int8_t lookup_table[] = {
        0, -1, 1, 0, 1, 0, 0, -1, -1, 0, 0, 1, 0, 1, -1, 0
    };

    while (true) {
        uint8_t current_state =
            (digitalRead(PIN_ENCODER_A) << 1) |
digitalRead(PIN_ENCODER_B);

        if (current_state != last_encoder_state) {
            int8_t direction = lookup_table[
                (last_encoder_state << 2) | current_state
            ];

            // Critical section for counter manipulation
            int8_t event = 0;
            portENTER_CRITICAL(&s_encoder_mux);
            encoder_counter += direction;
            if (encoder_counter >= INPUT_ENCODER_THRESHOLD) {
                event = 1;
                encoder_counter = 0;
            } else if (encoder_counter <= -INPUT_ENCODER_THRESHOLD) {
                event = -1;
                encoder_counter = 0;
            }
            portEXIT_CRITICAL(&s_encoder_mux);

            // Enqueue outside critical section
            if (event != 0) {
                encoder_enqueue(event);
            }
            last_encoder_state = current_state;
        }
        vTaskDelay(1);  // 1ms polling interval
    }
}
```

B.6 Button Debouncing with Multi-Event Detection

Source: *src/managers/input_manager.cpp*

Software debouncing implementation that detects single click, double click, and long press events. Uses a pending state machine to distinguish between click types within a configurable time window.

```
void input_manager_loop() {
    int reading = digitalRead(PIN_ENCODER_SW);

    // Debounce: reset timer on state change
    if (reading != last_button_state) {
        last_debounce_time = millis();
    }

    // Process stable state after debounce delay
    if ((millis() - last_debounce_time) > debounce_delay) {
        if (reading != button_stable_state) {
            button_stable_state = reading;

            if (button_stable_state == LOW) {  // Button pressed
                button_press_start_time = millis();
                unsigned long current_time = millis();

                // Double-click detection
                if (button_pending &&
                    (current_time - last_click_time) <
INPUT_DOUBLE_CLICK_INTERVAL_MS) {
                    button_double_clicked_flag = true;
                    button_pending = false;
                } else {
                    button_pending = true;
                    last_click_time = current_time;
                }
            }
        }
    }

    // Long press detection (while held)
    if (button_stable_state == LOW && button_press_start_time > 0) {
        if ((millis() - button_press_start_time) >=
INPUT_LONG_PRESS_THRESHOLD_MS) {
            button_long_pressed_flag = true;
```

```
            button_pending = false;
            button_press_start_time = 0;
        }
    }

    // Single click confirmation (after release + timeout)
    if (button_pending && button_stable_state == HIGH) {
        if ((millis() - last_click_time) >=
INPUT_DOUBLE_CLICK_INTERVAL_MS) {
            button_clicked_flag = true;
            button_pending = false;
        }
    }
    last_button_state = reading;
}
```

C. Bill of Materials

Component	Quantity	Unit Cost	Subtotal
ESP32-S3-DevKitC-1 N16R8	1	¥55	¥55
2.9" E-ink Display (GDEY029T94)	1	¥45	¥45
Capacitive Soil Moisture Sensor	1	¥8	¥8
Mini Water Pump + Tubing	1	¥15	¥15
Rotary Encoder with Button	1	¥5	¥5
3-Position Toggle Switch	1	¥3	¥3
P-MOSFET (AO3401A)	3	¥1	¥3
N-MOSFET (2N7002)	1	¥0.5	¥0.5
LiPo Battery 3.7V 2000mAh	1	¥25	¥25
TP4056 Charging Module	1	¥2	¥2
DC-DC Boost Converter (12V)	1	¥3	¥3

Resistors, Capacitors, Misc.	-	-	¥21.5
TOTAL			**¥186**

D. Prototype Photo

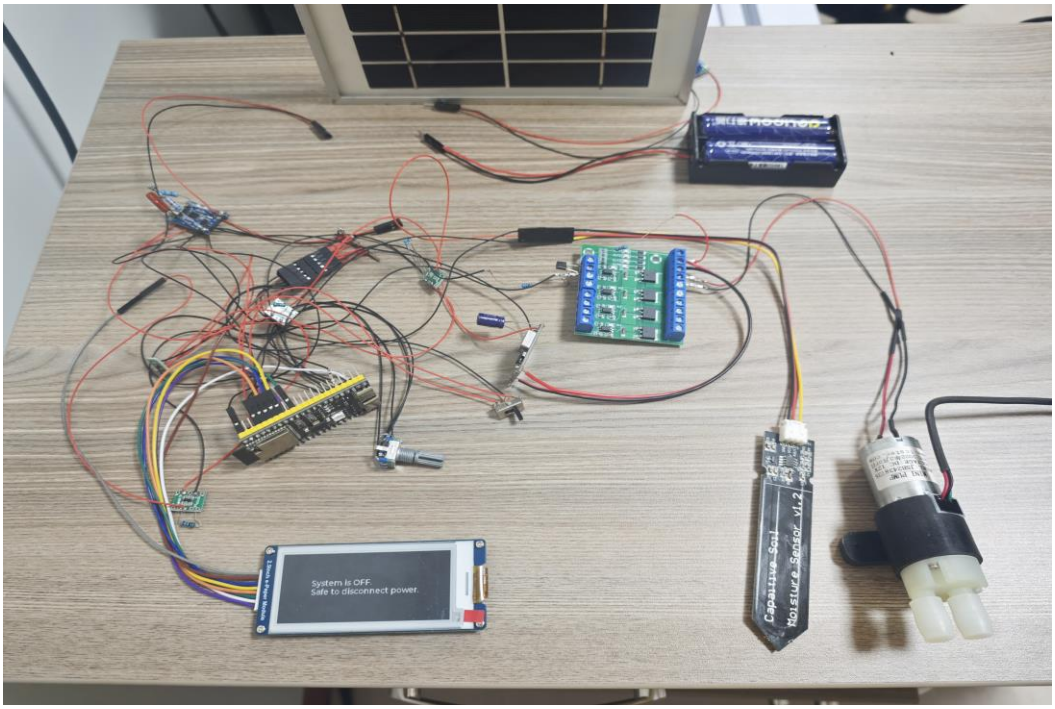

Figure C1: HydroSense Prototype